Upravljanje bazom podataka

Postoji dosta različitih sistema za upravljanje bazama podataka (MySQL, PostgreSQL...), ali s obzirom na to da Python podržava SQLite i da je biblioteka za rad sa tim sistemom deo njegovih ugrađenih biblioteka, u ovoj nastavnoj jedinici koristićemo SQLite. U ovoj nastavnoj jedinici obradićemo neke od osnova rada sa SQLite bazama, a detaljnija uputstva i informacije o bazama podataka i povezanim konceptima polaznik će pronaći u jednom od narednih kurseva.

Biblioteka sqlite3 se bazira na SQLite softveru koji je originalno napisan u C-u. Sam SQLite je baziran na SQL-u i razvijen je kako bi se on unapredio. SQL, skraćeno od Structured Query Language, jeste jezik pomoću kojeg se pristupa bazama podataka i njima upravlja. Sintaksički, ovaj jezik nudi naredbe za čitanje baze, brisanje, menjanje, kao i još mnogo toga.

Teorija o bazama

Baze podataka nam omogućavaju strukturirano i organizovano skladištenje podataka, kao i pouzdan i brz rad sa tim uskladištenim podacima. Baza je sačinjena od elemenata koji su predstavljeni u vidu tabela sa redovima i kolonama. Ove tabele se koriste za skladištenje podataka koje želimo da smestimo u bazu. Za svaku kolonu u tabeli se unapred određuje koji će tip podatka čuvati, dok se redovi u tabeli mogu definisati kao kolekcije povezanih vrednosti. Svaki red u tabeli se može obeležiti jedinstvenim identifikatorom – primarnim ključem.

Karakteristike SQLitea

Sam SQLite, nezavisno od svoje Python implementacije, nudi prednosti u odnosu na druge sisteme baza podataka koji se baziraju na SQL-u. Te prednosti su:

- SQLite ne zahteva poseban proces kako bi omogućio pristup fajlu, već pruža direktan pristup fajlu baze;
- SQLite ne zahteva prethodno dodatno konfigurisanje, pa se tako jednostavno i brzo može kreirati baza, za razliku od nekih drugih sistema baza podataka (npr. MySQL), za čije je korišćenje potrebno pokrenuti njihov server;
- kompatibilnost – pošto je čitava baza u jednom fajlu, a implementacija SQLitea je otvorenog koda, ovaj sistem upravljanja bazama podataka se može koristiti i na ostalim platformama i operativnim sistemima, a u poslednje vreme se najčešće može videti u razvoju Android aplikacija;
- Lite u SQLite nam govori da je reč o sistemu baza podataka koji je vrlo mali i ne zahteva prevelike resurse za pokretanje i održavanje.

Kreiranje baze

Pre nego što započnemo rad sa bazom, ako sam fajl baze ne postoji – potrebno ju je prvo kreirati. Ovo se može učiniti iz Pythona i biblioteke sqlite3. Takođe, za rad sa SQLite bazama podataka može se koristiti i alat sa grafičkim interfejsom – SQLite Browser.

Za kreiranje baze, prvo kreiramo objekat connection, koji će nam predstavljati konekciju sa bazom:

```
import sqlite3
conn = sqlite3.connect("TestDataBase.db")
```

Nakon izvršenja ove linije, u promenljivoj conn imamo objekat sqlite3.Connection, koji će u daljem kodu predstavljati našu konekciju sa bazom i sa njim možemo dalje upravljati bazom. Takođe, ako u trenutku izvršenja te linije fajl TestDataBase.db ne postoji – kreiraće se novi, a ako postoji istoimeni fajl – otvoriće se taj, već postojeći.

Pored korišćenja baze koja je trajno smeštena u fajlu, postoji i način kreiranja SQLite baze koja će postojati samo u RAM memoriji. Prednost ovakvog pristupa je to što olakšava i ubrzava testiranje određene željene funkcionalnosti bez potrebe za radom na samom fajlu. Mana je ta što nakon završetka našeg programa gubimo čitavu bazu sa svim podacima. Bazu smeštenu u memoriji ćemo kreirati linijom:

```
conn = sqlite3.connect(:memory:)
```

U našem primeru ćemo koristiti bazu podataka smeštenu u fajl TestDataBase.db.

Pošto smo uspostavili konekciju sa bazom, sledeće na redu je da omogućimo našem programu da upravlja podacima i tabelama iz nje. Ta funkcionalnost se postiže kursor (engl. cursor) objektom (deo klase sqlite3.Cursor). Kursor objekat nam omogućava pozive metoda koje mogu izvršavati naredbe SQLite jezika, preuzimanje podataka koje takvi upiti vraćaju itd. Objekat tipa sqllite3.Cursor se kreira korišćenjem metode .cursor() konekcijskog objekta:

```
cursor = conn.cursor().
```

Sada, pozivanjem metode .execute() nad kursor objektom možemo najpre kreirati prvu tabelu u praznoj bazi sledećom naredbom:

Primer kreiranja tabele u bazi TestDataBase.db

```
import sqlite3
conn = sqlite3.connect("TestDataBase.db")
cursor = conn.cursor()
cursor.execute("""CREATE TABLE IF NOT EXISTS Students(
   studentId INT PRIMARY KEY,
   name TEXT,
   age INT,
   grade INT);
""")
conn.commit()
```

Upišite kod u radno okruženje i isprobajte kako da napravite ovu bazu podataka:

Radno okruženje

Metodi .execute() se prosleđuje naredba tipa string, koja je zapravo napisana u SQL jeziku. Raščlanićemo je da bismo je lakše razumeti:

- CREATE TABLE – naredba za kreiranje tabele; obavezan je deo;
- IF NOT EXIST – uslov koji nam govori da se tabela kreira samo ako već ne postoji u bazi; korišćenje uslova je opciono, ali ako pokušamo da napravimo tabelu sa imenom koje već postoji u bazi – dobićemo grešku sqlite3.OperationalError: table Students already exists;
- Students – ime tabele koju kreiramo.

Dalje, u zagradama slede imena kolona koje želimo da kreiramo, kao i tip podataka koji će taj tip kolone sadržati. Više o podržanim tipovima se može pročitati u zvaničnoj dokumentaciji.

- (studentId INT PRIMARY KEY, – ime kolone studentId koja će predstavljati numerički identifikator studenta, tipa int, dok će ova kolona ujedno biti i primarni ključ tabele Students;
- name TEXT, – ime kolone name, koja će čuvati podatak o imenu, tipa tekst;
- age INT, – ime kolone age, koja će čuvati podatke o godištu, tipa int;
- grade INT); – ime kolone grade, koja će čuvati podatke u polu, tipa int;
- čitavu SQL naredbu završavamo znakom ;.

Na kraju, conn.commit() naredbom smo potvrdili i sačuvali ove izmene u bazi, odnosno izvršili SQL upit nad bazom. Bez ove naredbe, ako bismo prekinuli izvršavanje programa nakon cursor.execute() naredbe, promene koje smo napravili ne bi bile izvršene, odnosno, tabela ne bi bila napravljena.

Sada možete u radnom okruženju napraviti izmenu u kodu, tako što ćete dodati i podatak smer (string) koji predstavlja smer na koji je student upisan.

Pitanje

Koji tip podatka ćemo koristiti ako želimo da u kolonu smestimo celobrojne podatke kao što su na primer godine?

- TEXT
- **INT**
- PRIMARY KEY

Objašnjenje:

Tačan odgovor je da u SQLiteu koristimo INT tip podatka kada želimo da smestimo celobrojne podatke.

Dodavanje podataka u bazu

Dodavanje podataka u bazu se vrši takođe koristeći cursor objekat, kome se ovog puta prosleđuje SQL naredba tipa string i n-torka ili rečnik, u zavisnosti od toga da li je reč o parametrizovanom dodavanju ili parametrizovanom dodavanju sa imenovanim poljima. Naredba za ubacivanje elemenata u bazu koristeći parametrizovano dodavanje izgleda ovako:

```
student = (1, 'Karen', 25, 9)
cursor.execute("INSERT INTO Students VALUES (?, ?, ?, ?)", student)
conn.commit()
```

Značenje naredbe za dodavanje novog reda je:

- INSERT INTO Students – ovom naredbom SQL jezika određujemo u koju tabelu naše baze želimo da ubacimo podatke;
- VALUES (?, ?, ?, ?) – ovim delom naredbe, pomoću znakova pitanja ?, koji nam služe kao znakovi kojima govorimo SQL naredbi da će se na tim mestima proslediti elementi iz naše n-torke (plejsholder, engl. placeholder), ubacujemo elemente naše n-torke. Pa tako, ovi plejsholderi (znakovi pitanja) odgovaraju svakom elementu naše n-torke i koloni u tabeli. Ako bismo prosledili veći broj plejsholdera nego što je broj kolona u tabeli ili nego što je broj elemenata u n-torki, dobili bismo grešku.

Za ubacivanje podataka u bazu možemo koristiti i parametrizovano dodavanje sa imenima polja zajedno sa rečnicima umesto n-torke:

```
student = {"studentId":2, 'name':'Steven', "age":27, "grade":7}
cursor.execute("INSERT INTO Students VALUES (:studentId, :name, :age,
:grade)", student)
conn.commit()
```

Ova naredba se malo razlikuje od prethodne i to ponajviše u načinu kako određujemo kojim kolonama ćemo proslediti koje podatke. U ovom slučaju moramo ipak navesti imena kolona zajedno sa dve tačke ispred njihovih imena (:), za razliku od prethodnog načina, gde smo umesto imena koristili plejsholdere.

Sada možete u radnom okruženju dopuniti bazu podataka sa podacima o još jednom studentu koristeći execute metodu.

Metoda executemany()

Kroz ove primere smo uvideli kako se po jedan red ubacuje u bazu. Ako želimo odjednom da ubacimo više redova, umesto cursor.execute() koristićemo cursor.executemany() metodu. Ovoj metodi takođe kao prvi parametar prosleđujemo string koji predstavlja SQL naredbu, a za drugi parametar umesto n-torke ili rečnika (u zavisnosti od toga koju varijantu koristimo), prosleđujemo ili listu n-torki ili listu rečnika:

Primer parametrizovanog dodavanja

```
student = [(1, 'Karen', 25, 9), (2, 'Steven', 27, 9)]
cursor.executemany("INSERT INTO Students VALUES (?, ?, ?, ?)",
student)
```

Primer parametrizovanog dodavanja sa imenima polja

```
student = [{"studentId":2, 'name':'Steven', "age":27, "grade":7},
           {"studentId":1, 'name':'Karen', "age":25, "grade":9}]
cursor.executemany("INSERT INTO Students VALUES (:studentId, :name,
:age, :grade)", student)
```

Dopunite u radnom okruženju bazu podataka sa podacima o još dva studenta koristeći executemany metodu.

Čitanje iz baze podataka

Do sada smo naučili kako se podaci smeštaju u tabelu; međutim, pored upisa u bazu, potrebno nam je i čitanje iz baze, odnosno tabele. Iščitavanje se vrši koristeći takođe execute() metodu nad kursor objektom:

Primer iščitavanja podataka iz baze

```
cursor.execute("SELECT * FROM Students;")
all_results = cursor.fetchall()
print(all_results)
```

Pa bi ceo kod izgledao ovako:

Radno okruženje

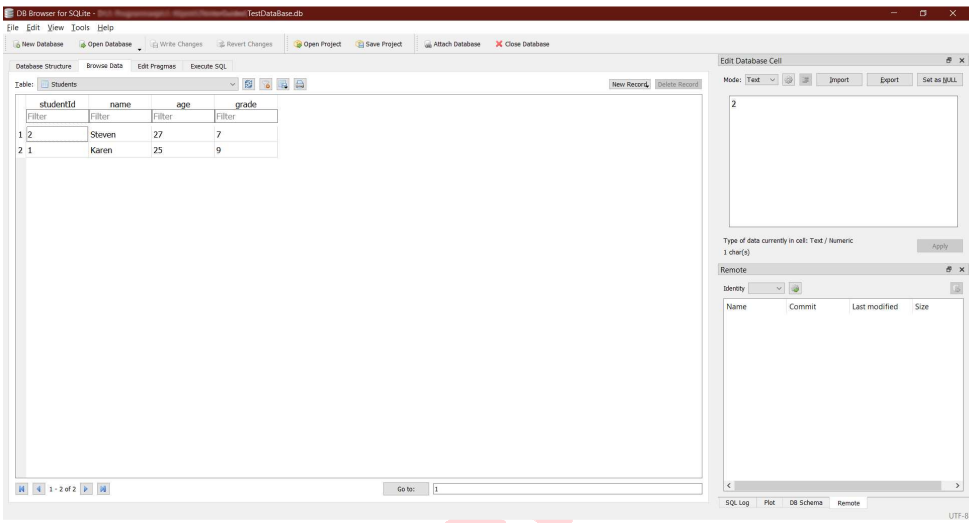
```
import sqlite3
conn = sqlite3.connect("TestDataBase.db")
cursor = conn.cursor()
cursor.execute("""CREATE TABLE IF NOT EXISTS Students(
   studentId INT PRIMARY KEY,
   name TEXT,
   age INT,
   grade INT);
""")
conn.commit()

student = [{"studentId":2, 'name':'Steven', "age":27, "grade":7},
    {"studentId":1, 'name':'Karen', "age":25, "grade":9}]
cursor.executemany("INSERT INTO Students VALUES
(:studentId, :name, :age, :grade)", student)


cursor.execute("SELECT * FROM Students;")
all_results = cursor.fetchall()
print(all_results)
```

SQL naredba za čitanje uvek počinje SELECT ključnom rečju, nakon koje prosleđujemo tačna imena kolona koje želimo da pročitamo ili, u našem slučaju, džokerski znak zvezdicu (*), kojim preciziramo da želimo sve kolone. FROM Students deo naredbe govori o tome iz koje tabele preuzeti prethodne kolone. Dakle, ovom naredbom smo doslovno rekli: „obeležiti sve kolone iz tabele Students i vratiti podatke u program”. Ako postoje redovi koji ispunjavaju ovakav uslov – ta naredba će nam vratiti te redove.

Nakon što se naredba izvršila, rezultat je uskladišten u kursor objektu. Za pristup tim rezultatima možemo koristiti više metoda:

- fetchone() – vraća samo jedan red kao rezultat, iako u tabeli može biti više redova; povratna vrednost ove metode je n-torka;
- fetchmany(n) – parametar n označava koliko zapravo redova želimo da vratimo; povratna vrednost ove metode je lista n-torki koje predstavljaju sve redove koji odgovaraju uslovima zadatim našom SQLite naredbom;
- fetchall() – vraća sve redove koji odgovaraju zadatom uslovu iz SQLite naredbe.

U našem primeru smo se opredelili za fetchall(), koja vraća listu redova koji odgovaraju zadatom uslovu iz SQLite naredbe.



Slika 4.1. Grafički prikaz naše tabele u slučaju korišćenja programa DB Browser for SQLite

Izmenite kod u radnom okruženju tako da napravite bazu sa imenom Knjige u kojoj je potrebno da definišete podatke id_knjige (ceo broj), ime (string), autor (string), godina_izdanja (ceo broj). Unesite podatke za 2 knjige u bazu podataka koristeći executemany metode, takođe ispišite podatke iz baze na standardni izlaz.

Radno okruženje

Napomena:

Za rad sa bazama podataka važi jedno pravilo koje važi i za rad sa fajlovima. Iako sam Python vodi računa o zatvaranju fajlova i konekcija sa bazama podataka na kraju izvršenja programa – dobra je praksa da ih ipak mi sami zatvorimo nakon što završimo rad sa njima. Kursor objekat zatvaramo sa cursor.close(), dok objekat koji predstavlja konekciju sa bazom zatvaramo sa conn.close(). Nakon što ugasimo kursorski objekat, svi podaci koje on sadrži postaju nedostupni, pa tako, ako smo pokrenuli naredbu SELECT * FROM… koristeći kursor i ugasili kursor, onda nam podaci vraćeni tom naredbom nisu više dostupni.

Rezime

- Biblioteka sqlite3 se bazira na SQLite softveru, koji je originalno napisan u C-u. Sam SQLite je baziran na SQL-u i razvijen je kako bi se on unapredio.
- Baze podataka nam omogućavaju strukturirano i organizovano skladištenje podataka, kao i pouzdan i brz rad sa tim uskladištenim podacima.
- Baza je sačinjena od elemenata koji su predstavljeni u vidu tabela sa redovima i kolonama. Ove tabele se koriste za skladištenje podataka koje želimo da smestimo u bazu. Za svaku kolonu u tabeli se unapred određuje koji će tip podatka čuvati, dok se redovi u tabeli mogu definisati kao kolekcije povezanih vrednosti. Svaki red u tabeli se može obeležiti jedinstvenim identifikatorom – primarnim ključem.
- Za kreiranje baze, prvo kreiramo objekat connection, koji će nam predstavljati konekciju sa bazom.
- Kursor objekat nam omogućava pozive metoda koje mogu izvršavati naredbe SQLite jezika, preuzimanje podataka koje takvi upiti vraćaju itd.
- Metodi .execute() se prosleđuje naredba tipa string, koja je zapravo napisana u SQL jeziku.
- Dodavanje podataka u bazu se vrši takođe koristeći cursor objekat, kome se ovog puta prosleđuje SQL naredba tipa string i n-torka ili rečnik, u zavisnosti od toga da li je reč o parametrizovanom dodavanju ili parametrizovanom dodavanju sa imenovanim poljima.
- Ako želimo odjednom da ubacimo više redova, umesto cursor.execute() koristićemo cursor.
- Metode fetchone(), fetchmany() i fetchall nam služe za dobavljanje rezultata samog upita:
    - fetchone() – vraća kao rezultat samo jedan red, iako u tabeli može biti više redova; povratna vrednost ove metode je n-torka;
    - fetchmany(n) – parametar n označava koliko zapravo redova želimo da vratimo; povratna vrednost ove metode je lista n-torki koje predstavljaju sve redove koji odgovaraju uslovima zadatim našom SQLite naredbom;
    - fetchall() – vraća sve redove koji odgovaraju zadatom uslovu iz SQLite naredbe.